μJSON, a Lightweight Compression Scheme for Embedded GNSS Data Transmission on IoT Nodes

Sérgio D. Correia*†, *Member, IEEE,* Rafael Perez*, João Matos-Carvalho† and Valderi R. Q. Leithardt*†, *Member, IEEE.*

*VALORIZA, Polytechnic Institute of Portalegre, 7300-555 Portalegre, Portugal

†COPELABS, ULHT, Campo Grande 376, 1749-024 Lisboa, Portugal

Correspondence: scorreia@ipportalegre.pt

***Abstract*—Embedded computing is a sector in strong growth, driven by the increasing offer in the internet of things. Since position information is typically one of the intrinsic features of a sensory platform, the present work considers and analyzes the embedded lossless compression of localization data. A structural data scheme is proposed based on the number of occurrences of the independent coordinate components, named as μJSON. It is shown that, when considering a low-level language implementation on an embedded processor acquiring GNSS data, the proposed schema implies lower compression rates than its counterparts. An embedded test-bench platform is assembled, and several real case scenarios are considered for effectiveness and validation purposes.**

***Index Terms*—Data Transmission, Embedded Computing, Global Navigation Satellite System, Internet of Things, Lossless Compression**

I. Introduction

The Internet of Things (IoT) paradigm has been receiving enormous attention recently, penetrating increasingly daily tasks. The term IoT generally applies to scenarios where network connectivity and computing capability extend to objects, sensors, and all sorts of items. Those are not commonly considered computers, allowing these items to generate, exchange, and consume information data with none or minimal human intervention [1], [2]. Application examples goes from agriculture [3], home automation [4], energy management [5]–[7], healthcare [8] or industry [9].

A critical factor in the development of the IoT is the communication capabilities of sensor/actuator nodes of the infrastructure. Computing paradigms such as Cloud, Fog, and Edge computing, can be considered as the backbone of different services like data offloading, resource and device management, processing stages, or user interfaces [10]. Since most of the data is originated on those sensors/actuators, and considering that a large set of these devices can be battery-operated, power depletion and techniques to extend the battery life becomes an imperative factor [11]. Although each sub-task such as sensing, processing, and data transmission consumes energy, previous studies have demonstrated that network data transmission causes a significant cut of the overall energy consumption of battery-powered devices [12]–[14]. To this end, the use of data compression techniques as long be considered for bandwidth optimization [17], and more recently, for energy saving, shortening the transmission time [15].

Data compression intends to minimize the size of the data packet to be stored and transmitted by reducing the size of some data files, either by using lossless or lossy compression methods [16]. While sensory data may be considered with some losses, localization information is less susceptible to data loss since it could profoundly tamper with the positioning feature [17]. Therefore, lossy compression would be adequate in cases where the discard of data information would not be perceptible (such as audio, image or video) [18]. However, compression methods usually have poor performance in terms of space-time complexity or compression rate when considered on embedded sensors, leading to a rapid depletion of memory, storage, computing, and energy resources [19]. Nonetheless, balancing the need for complex mathematical calculation for high-performance data compression algorithms and the consequently obtained gain is often a challenging task [20].

The above discussion leads to a new methodology for data packaging and subsequent compression and transmission. Globally, the new compression technique consists of two different stages. A first pre-processing stage encapsulates the data in a JavaScript Object Notation format, avoiding Global Navigation Satellite System (GNSS) data repetition and performing the first compression level (here identified as μJSON format for trajectory representation). Secondly, some lossless compression scheme is applied before the data is transmitted through some communication channel. To allow for a greater generalization of the methodology, methods based on coding by repeated sequences, and methods based on the character commutative frequency, are both implemented, simulated, and tested.

An embedded test-bench is planned and implemented to demonstrate the proposed methodology's repeatability, effectiveness, and reliability, besides simulation results. Based on the developed hardware, several real-case scenarios are considered with the acquisition of real GNSS data on different trajectory paths. Both simulation and real-life results show how the new method stands out compared to alternative solutions for a wide range of conditions concerning the number of acquired coordinates, traveled distance, and displacement.

The remaining paper is organized as follows. Section II describes the proposed method and Section III an hardware test-bench for real-time data acquisition and compression. Section IV validates and assesses the method's performance

2022 5th Conference on Cloud and Internet of Things (CIoT) | 978-1-6654-7964-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/CIOT53061.2022.9766635

and finally, Section V concludes the paper.

II. μJSON for Trajectory Data Representation

The proposed method consists of two independent stages. The first stage complies with formatting the localization data into some μJSON format, using the JavaScript Object Notation (JSON) formalism, but avoiding the repetition of some coordinates fields. When considering GNSS coordinates in the DMS (degrees, minutes, seconds) format, and depending on the point of the earth, one can travel a significant distance without stepping onto a new latitude or longitude. The same reasoning goes for the minutes' field, but with a lower traveled distance. As such, each data packet includes a timestamp (TS in Figure 1), the number of samples (NS), and the number of constant fields (FX), besides each DMS field for the latitude and the longitude (PL, DL, and SL for the latitude, and PN, DN, MN, and SN for the longitude).

```
{
    "TS": int,           //Timestamp
    "NS": short int,     //Num. Samples
    "FX": byte,          //Fixed Fields
    "lat": {             //Latitude
        "PL":[byte],     //Signal
        "DL":[byte],     //Degrees
        "ML":[byte],     //Minutes
        "SL":[int fixed point]
        },               //Milliseconds
    "lng": {             //Longitude
        "PN":[byte],     //Signal
        "DN":[byte],     //Degrees
        "MN":[byte],     //Minutes
        "SN":[int fixed point]
        }                //Milliseconds
}
```

Fig. 1: μJSON Format for Trajectory Representation

One field retains its value in the case of one fixed value ($FX = 1$). In this case, it will be the signal field PL and PN. Similarly, when $FX = 2$, the most significant numeric field DL and DN have only one value, and the others (ML and SL, MN and SN) will be a vector of size NS. In the case of $FX = 3$, both degrees and minutes will be constant, and the less significant field will be an array of size NS. In this first processing stage, sequences in which the same data value occurs in many consecutive data elements are stored as a single data value and count rather than as the original form. This is the basic principle of the Run-Lenght Encoding (RLE) method [21]. Thus, some degree of compression is expected when applying the described scheme for data storage, although maintaining a structured data format due to the JSON formalism. Also, this processing stage is of linear complexity due to the possibility of an in-line calculation since only sequential data is analyzed. Depending on the embedded processor used for the data acquisition, the GNSS data format may be stored in binary and only be put in the μJSON format when transmitted to some data server. However, μJSON brings data portability and interruptibility. The μJSON schema deals with the unbalanced data, allowing a more flattened distribution of the frequencies of the Huffman dictionary. The usual technique to deal with unbalanced data cannot be applied due to the restriction that the decompression must not lose data and does must create an exact copy of the compressed information [22].

Following the first or pre-processing stage, and only when data transmission occurs, a second stage performs a formal compression of the data sent over some communication channel. This final step will comprise implementing some low complexity compression algorithm, where the input consists of the μJSON schema developed. At this stage, one candidate algorithm will be considered due to its simplicity and efficiency, that is, Huffman Coding [23]. It is essential to remember that the compression algorithm complexity is crucial when considering that the context implementation is an IoT node. Thus, the computer system will probably comply with a data bus of 8 or 16 bits, some simple computational architecture without a math co-processor or a floating-point unit.

The Huffman model uses the probability of occurrence of a character. Therefore, the Huffman method should not be compromised with the μJSON schema since the individual probability is affected but not the actual likelihood of the frequency event.

III. Developed Test-Bench for Real-time Acquisition and Compression

The testing procedure of software algorithms is usually implemented with high-level programming languages on high-performance computing hardware. Nevertheless, when considering IoT frameworks and embedded computing hardware, several limitations arise, and other testing procedures are needed. To validate the proposed method on its final use context, and an embedded test-bench was developed to perform several case studies with real-life data sources and an IoT hardware target.

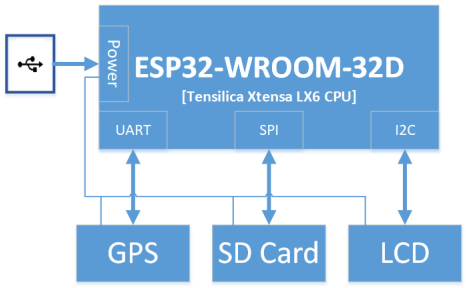
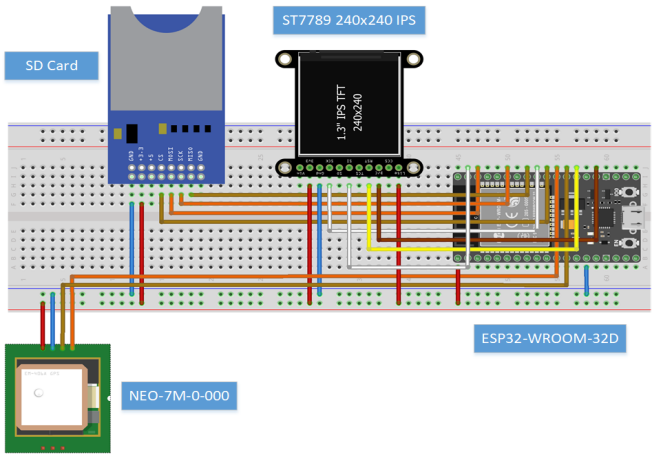
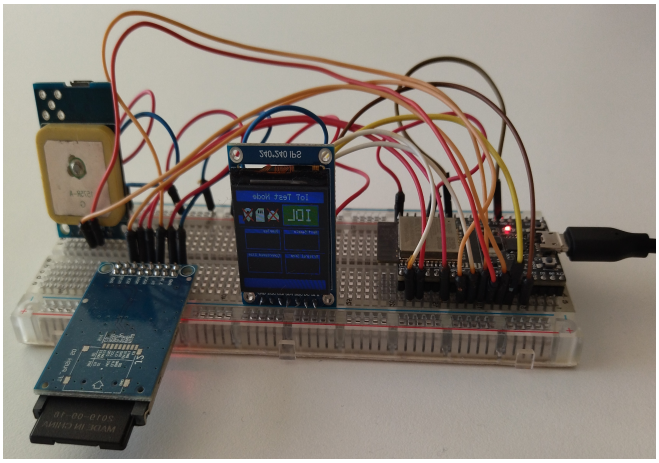


Fig. 2: Block Diagram of the Prototype Components



(a) Connection Schematics of the Developed Prototype



(b) Prototype Assembled on a Bread Board

Fig. 3: Final Fully Functional Prototype

To this end, an ESP32 (ESP32-WROOM-32D) module from Espressif[1] was considered. The hardware module consists of an LX6 32-bit RISC core processor from Tensilica Xtensa, featuring 512 KBytes of RAM and 4 MBytes of flash memory. The ESP32 hardware also incorporates a native 2.4GHz WIFI module implementing norms 802.11 b/g/n with an internal TCP/IP (IPv4) stack. It features several GPIO (general-purpose input/output) pins, analog-to-digital converters, and multiple communication interfaces. The processor incorporates several different low power modes with small energy consumption's, making it very suitable and affordable modules for IoT projects. The module presents itself with an internal power regulator and a USB chipset. It provides a flexible connection to a personal computer for programming purposes and allows powering different peripheral hardware modules. Besides the ESP32, the full assembled test-bench also includes a NEO-7M-0 u-blox[2] GNSS module to acquire real-time coordinates, a 240×240 TFT display for debugging and user interface purposes, and an SD card to store all sort of data of the different testing stages, all powered from the ESP32 module (Figure 2). Due to the different feature of each peripheral module, the GNSS exchange information with the ESP32 through a TTL full-duplex universal asynchronous receiver-transmitter (UART), the SD card uses a Serial Peripheral Interface (SPI) bus, and the TFT display an Inter-Integrated Circuit (I2C) communication bus as shown in the block diagram of Figure 2. This means that all peripheral will have different Operating System (OS) drivers.

When considering embedded software testing techniques, one can distinguish between "White-Box" and "Black-Box" methodologies, depending on whenever the explicit knowledge of the implementation details is under analysis or not [24]–[26]. On the one hand, the principle of a "Black-Box" test implies feeding the system with some inputs, and the resulting outputs are analyzed as to whether it complies with the expected behavior. On the other hand, a "White-Box" technique would imply the knowledge of the firmware internal structure [24], [26]. Given our primary focus and the fact that our platform's major concern is to generate a compressed set of coordinates with a two-stage processing strategy, a Black-Box approach is considered here. To this end, several different trajectory paths will be generated, either by simulation where the coordinate point is obtained using some tool or by real-life GNSS signal acquisition.

The complete prototype was assembled on a breadboard. Figure 3 shows the connection schematic of the processor with all peripheral modules (left) and the complete stacked prototype on the right. For the user application control and debugging purposes, there are two interfaces: a TFT showing several information's and a web page that, besides reporting information, also allow control inputs.

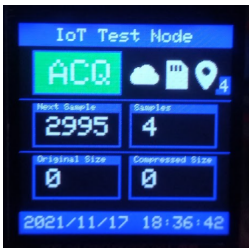


Fig. 4: TFT Layout Screen when in Acquiring Mode

Concerning the TFT display (Figure 4) the following information is displayed: the state of the prototype that can be IDL-Standby, ACQ-Acquiring, BUF-Buffering, and ERR-Error; the connections status of the WiFi, the SD card, and the GSP module where the is also information on the number of visible satellites; the remaining time (in milliseconds) for the subsequent sample acquisition; and the number of acquired samples; and finally the size of the data packet before and after being compressed. Also, there is a status bar at the

[1]https://www.espressif.com/en/products/socs/esp32
[2]https://www.u-blox.com/en/product/neo-7-series

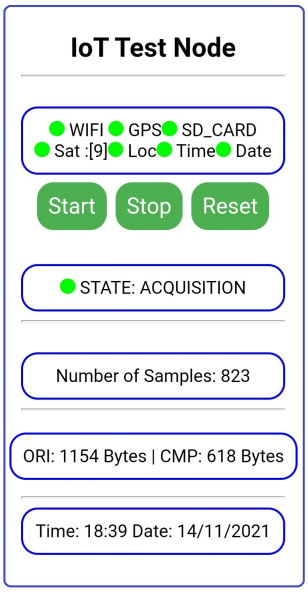


Fig. 5: Smart Phone Screen Capture of the WEB-Page for Control and Management

bottom with the date and time supplied by the GPS module. All this information allows acknowledging any error with the peripherals, data overflow, and the final compression rate obtained with the current data acquisition packet.

The second user interface, which also allows user control, is a web page stored on the ESP32. More precisely, the device can answer an HTTP request and, after opening a browser and navigating to the ESP32 IP's address, the processor responds with just enough HTML for a browser to display the same information as the TFT, with additional controls in the form of three buttons (see Figure 5).

A simple state machine is designed and implemented to manage all peripherals and the data flow. As it can be seen from Figure 6, there are four states, the same ones that were discussed for presentation in the TFT display. The system starts up in a state of IDL-Standby, where it remains until the user sets the *Start Button* through the web page. From this

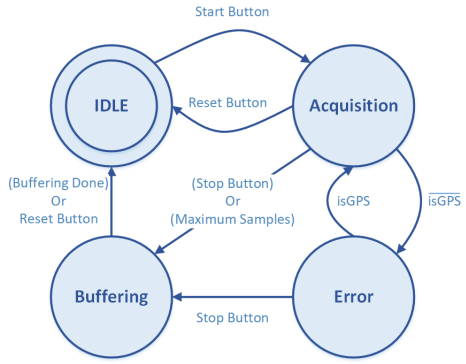


Fig. 6: Firmware State Machine for Process Control

point on, the system starts acquiring signals from the GPS module and stores them on the RAM, respecting a trigger with a stipulated time interval for acquisitions. Suppose some *Error* is detected concerning the GPS, such as a communication failure or a minimum number of satellites that do not allow acquiring GPS coordinates, the system enters an *Error* state. If the GPS conditions are restored, the acquisition restarts from the previous point without losing data. When in the *Error* state, the *Stop Button* returns to the *Idle* state saving the data already acquired. The system stops automatically if the maximum number of samples is reached, or the *Stop Button* is pressed, always saving the data onto the SD card.

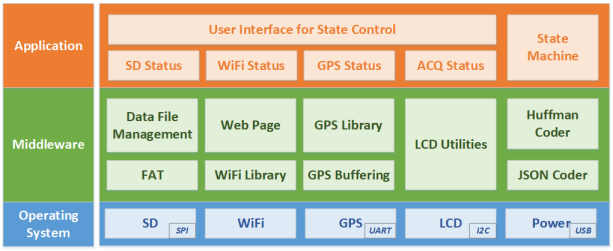


Fig. 7: Firmware Layer Model

In short, the firmware is assumed as a standard three layers model, divided between Operating System, Middleware, and the Application Layer. Figure 7 gives a detailed overview of each component present in each layer. The operating system complies with a block for each communication interface that deals with the different peripherals. The middleware controls the data flow with read/write operations on the SD Card, the web-server to display the web page, the GPS NMEA parsing, the TFT utilities, and the project-specific libraries that deal with the μJSON proposed coder, and the Huffman coder. Finally, the application layer deals with state machine control, user interface, and internal status.

IV. Results and Discussion

The process to demonstrate the proposed methodology's validation, effectiveness, and reliability was considered with two different steps. Firstly, several coordinates, simulating different trajectories paths, were created with the Google Earth tool[3]. Then, a Python[4] script generated the NMEA strings that the GPS module would supply. This way, simulation's scenarios could be fed to the μJSON and Huffman coders. Secondly, the developed prototype was used to acquire real GPS data, code the μJSON format, and the Huffman compression stage.

A. Simulations

To accommodate different shapes of trajectory paths, and with varying sizes in terms of the number of coordinates, the simulations considered an almost straight trajectory with 872 m long and 50 points, represented in red in Figure 8 and identified as the number 1. Also, a trajectory with 100

[3]https://www.google.com/intl/pt-PT/earth/
[4]https://www.python.org/

coordinates points and a traveled distance of 2.610 km, colored blue in Figure 8 and identified with the number 2. Finally, a longer trajectory path, with 200 coordinate points, a traveled distance of 6.100 km, is marked in green and identified as the number 3 in Figure 8.

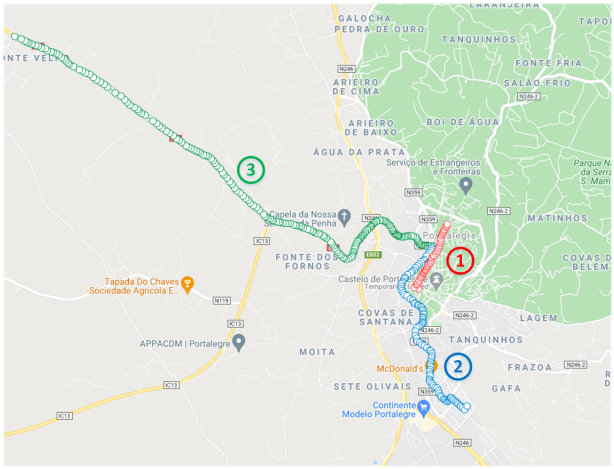


Fig. 8: Simulation Trajectory Paths (Generated with Google Maps™, owned by Google LLC)

Although it is expected that Huffman coding provides the best compression rate, and to validate this hypothesis, both LZ77 [27] and LZW [28] methods are considered for comparison. Also, it is mandatory to state that, in the case of the Huffman method and when calculating the compression rate, the Huffman dictionary is also considered as transmitted information, as a header of the compressed data. While some dictionaries are sometimes regarded as known (language rates) [29], [30], in this case, the dictionary is built among the character frequencies that are calculated for each packet transmission.

Id.	Huff.	uJSON	Huff.(uJSON)	uJSON+Huff.
1	**58%**	66%	68%	**45%**
2	**50%**	64%	64%	**36%**
3	**45%**	63%	50%	**32%**
mean	**51%**	64%	61%	**38%**
std	**5%**	1%	8%	**5%**

TABLE I: Simulations results applying Huffman Coding

Id.	LZ77	uJSON	LZ77(uJSON)	uJSON+LZ77
1	**54%**	66%	72%	**47%**
2	**48%**	64%	63%	**41%**
3	**43%**	63%	57%	**36%**
mean	**48%**	64%	64%	**41%**
std	**4%**	1%	6%	**4%**

TABLE II: Simulations results applying LZ77 Method

Tables I, II and III shows the compression rates obtained with Huffman Coding, the LZ77, and LZW methods respectively. Considering these tables, the first column identifies the trajectory from Figure 8, the second column is the compression ratio when simply applying the method to the raw data

Id.	LZW	uJSON	LZW(uJSON)	uJSON+LW
1	**69%**	66%	88%	**58%**
2	**58%**	64%	77%	**49%**
3	**49%**	63%	68%	**43%**
mean	**59%**	64%	78%	**50%**
std	**8%**	1%	8%	**6%**

TABLE III: Simulations results applying LZW Method

without the μJSON encapsulation, and corresponding to each table (Huffman, LZ77 or LZW). The third column is equal on all three tables. It corresponds to the compression ratio of implementing the proposed schema on the original raw data, the μJSON format (pre-processing stage not dependent on the coding algorithm). The following column concerns the compression ratio of the corresponding method applied on the μJSON format. Finally, the last column is the total compression ratio, the relation between the original raw data size and the compressed μJSON.

From the obtained results, one can see that:

1) Only by applying the μJSON format, a $2/3$ compression rate is obtained without compromising the sparsity of the data, enabling a new compression stage.
2) Regardless of the coding method, firstly applying the μJSON encapsulation always implies getting lower compression ratios regarding the compression method itself.
3) When considering the overall compression rate, combining the μJSON format with Huffman coding always archives the lowest value (for the three trajectory paths).
4) Increasing data packet size (paths with more coordinate points), the compression ratio improves.

The initial hypothesis is validated for different trajectory paths, where a two-stage compression schema archives good performances, complying with the flexibility, interruptibility, and structured data format of JSON.

B. Acquired Data

The second group of results concerns real trajectory data that is acquired with the developed test-bench platform. Several different trajectory paths are recorded on the SD card using the web interface and monitored through the TFT display. For debugging purposes, the test-bench saves on the SD card a KML[5] file with all coordinates acquired, a JSON file that implements the μJSON format, and a HUF file that contains the compressed data. All files are stored on directories that consider the time/date on its structure, allowing several different tests to be pursued without changing or removing the card. On a second step, the SD card is removed, some Python script decompresses the HUF file, following a reconstruction of the KML format based on the μJSON. The obtained KML is then compared with the saved file on the card, where a perfect match implies the success of all procedures. Also, the compression ratio can be evaluated based on the saved data files.

The total test data comprises six trajectories, with a different number of coordinate points and different displacements. Table

[5]https://www.ogc.org/standards/kml

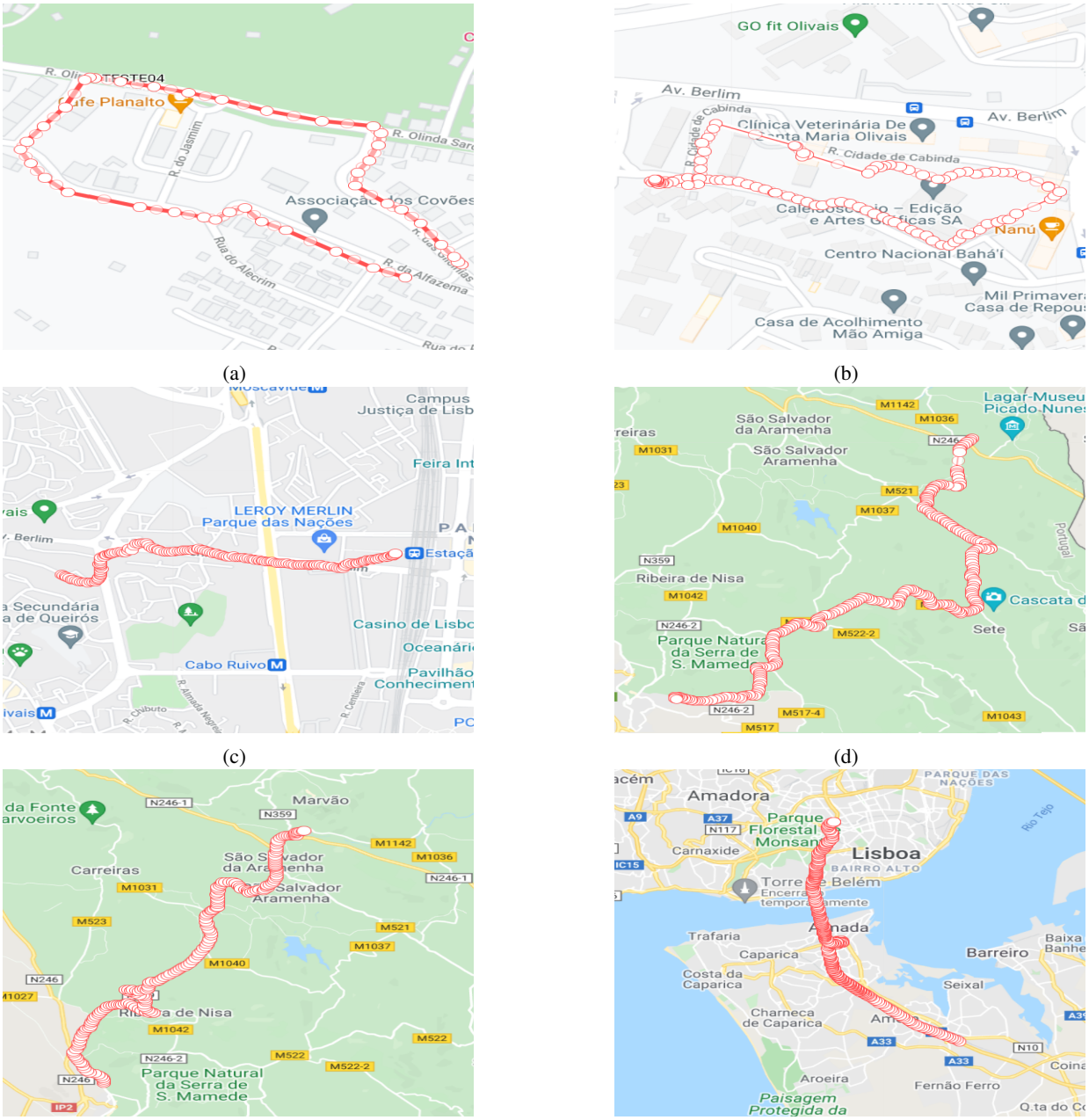


Fig. 9: **Trajectories acquired with the developed test-bench** (Generated with Google Maps™, owned by Google LLC).

IV resumes the trajectories characterization and the obtained compression rates.

Id	Ns	ΔS(m)	S(m)	Original	Comp.	Ratio
(a)	41	26	823	771	478	62%
(b)	120	6	748	1 201	654	54%
(c)	170	1 000	1 250	2454	1 113	45%
(d)	200	11 220	14 500	3 316	1 554	47%
(e)	277	11 500	18 700	4 548	2 068	45%
(f)	478	16 000	18 300	7 764	3 395	44%

TABLE IV: Characterization and Resumed Test Results

The considered datasets goes from small trajectories with 41, 120 or 170 number of coordinates points (N_S), and bigger ones, with 200, 277 and 478 coordinate points; from a travelled displacement ($S(m)$) of $823m$, up to approximately $18km$. It is also considered the total displacement ($\Delta S(m)$), that is, the distance between the initial and final coordinate points.

It is worth mentioning that all tests passed the validation performed; all the backward data reconstruction matched the KML saved files. From the results presented in Table IV, one can see that the obtained results are in line with what was announced in the simulations. While Figure 9 (a) with only 41 coordinate points implies a compression ration of 64%, Figure

9 (b) that triples the number of coordinate points, corresponds to a compression rate of 54%. Figures 9 (d), (e), and (f), with a number of coordinate points higher than ten thousand, all archived compression rates of around 40%. As previously mentioned, when dealing with simulation coordinate points, the compression rates of data packets with a higher number of coordinates points imply lower compression rates.

V. Conclusion

The present work proposed a new scheme for embedded trajectory compression that uses newly μJSON format for data encapsulation. It is shown that, when considering several different trajectory typologies, the combination of μJSON with a lossless compression method always Leeds to lower compression rates. An embedded test-bench was also developed to validate and demonstrate the methodology's effectiveness and reliability, including hardware and firmware. Real case scenarios also showed that lower compression rates could be archived when applying the new method, making it suitable for IoT sensor nodes where localization information is needed. Even though a few characters are added to build the JSON standard, the overall compression rate does not lose concerning direct compression on the trajectory coordinates. The advantage of interruptibility, portability, and ease of interpretation is added.

As for future work, a new platform to apply the methodology on sensor drones is under development.

Acknowledgment

This work was supported in part by the Fundação para a Ciência e a Tecnologia, under project UIDB/05064/2020 and UIDB/04111/2020; and also ILIND-Instituto Lusófono de Investigação e Desenvolvimento, under project COFAC/ILIND/COPELABS/1/2020 and FAC/ILIND/COPELABS/3/2020.

References

[1] Zaheeruddin and H. Gupta, "Foundation of iot: An overview," *Internet of Things (IoT)*, p. 3–24, 2020.

[2] H. Shariatmadari, S. Iraji, and R. Jäntti, *From Machine-to-Machine Communications to Internet of Things: Enabling Communication Technologies*. Chapman and Hall/CRC, 2017.

[3] M. R. M. Kassim, "Iot applications in smart agriculture: Issues and challenges," in *2020 IEEE Conference on Open Systems (ICOS)*, 2020, pp. 19–24.

[4] S. K. Vishwakarma, P. Upadhyaya, B. Kumari, and A. K. Mishra, "Smart energy efficient home automation system using iot," in *2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU)*, 2019, pp. 1–4.

[5] E. M. Pinheiro and S. D. Correia, "Software model for a low-cost, iot oriented energy monitoring platform," *International Journal of Computer Science and Engineering*, vol. 5, no. 7, p. 1–5, 2018.

[6] E. M. Pinheiro and S. D. C. Correia, "Hardware architecture of a low-cost scalable energy monitor system," *International Journal of Engineering Trends and Technology*, vol. 61, no. 1, p. 1–5, 2018.

[7] E. M. Pinheiro and S. D. Correia, "Analog input expansion board based on i2c communication with plug-and-play feature, applied to current measurements," *International Journal of Electronics and Communication Engineering*, vol. 5, no. 9, p. 1–5, 2018.

[8] I. S. G. Brites, L. M. da Silva, J. L. V. Barbosa, S. J. Rigo, S. D. Correia, and V. R. Q. Leithardt, "Machine learning and iot applied to cardiovascular diseases identification through heart sounds: A literature review," *Informatics*, vol. 8, no. 4, p. 73, 2021.

[9] L. G. Seng, K. L. K. Wei, and S. J. Narciso, "Effective industry ready iot applied courseware - teaching iot design and validation," in *2020 IEEE Global Engineering Education Conference (EDUCON)*, 2020, pp. 1579–1583.

[10] S. N. Swamy and S. R. Kota, "An empirical study on system level aspects of internet of things (iot)," *IEEE Access*, vol. 8, pp. 188 082–188 134, 2020.

[11] A. Elhaddad, H. Bruckmeyer, M. Hertlein, and G. Fischer, "Energy consumption evaluation of cellular narrowband internet of things (nb-iot) modules," *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, 2020.

[12] R. K. Singh, P. P. Puluckul, R. Berkvens, and M. Weyn, "Energy consumption analysis of lpwan technologies and lifetime estimation for iot application," *Sensors*, vol. 20, no. 17, 2020.

[13] Nasaruddin, M. Andriani, Melinda, and M. Irhamsyah, "Analysis of energy efficiency for wi-fi 802.11b multi-hop networks," in *2013 IEEE International Conference on Communication, Networks and Satellite (COMNETSAT)*, 2013, pp. 64–68.

[14] L. Guegan and A.-C. Orgerie, "Estimating the end-to-end energy consumption of low-bandwidth iot applications for wifi devices," in *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2019, pp. 287–294.

[15] R. Das, A. K. Mishra, C. Nicopoulos, D. Park, V. Narayanan, R. Iyer, M. S. Yousif, and C. R. Das, "Performance and power optimization through data compression in network-on-chip architectures," in *2008 IEEE 14th International Symposium on High Performance Computer Architecture*, 2008, pp. 215–225.

[16] A. Gopinath and M. Ravisankar, "Comparison of lossless data compression techniques," in *2020 International Conference on Inventive Computation Technologies (ICICT)*, 2020, pp. 628–633.

[17] C. Chen, D. Zhang, Y. Wang, and H. Huang, *Trajectory Data Compression*. Springer Singapore, 2021, pp. 25–46.

[18] L. Gavalakis and I. Kontoyiannis, "Fundamental limits of lossless data compression with side information," *IEEE Transactions on Information Theory*, vol. 67, no. 5, pp. 2680–2692, 2021.

[19] L. Benini, D. Bruni, A. Macii, and E. Macii, "Hardware-assisted data compression for energy minimization in systems with embedded processors," in *Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition*, 2002, pp. 449–453.

[20] A. Gopinath and M. Ravisankar, "Comparison of lossless data compression techniques," in *2020 International Conference on Inventive Computation Technologies (ICICT)*, 2020, pp. 628–633.

[21] D. Salomon, *Data compression the complete reference*. Springer, 1998.

[22] M. Khushi, K. Shaukat, T. M. Alam, I. A. Hameed, S. Uddin, S. Luo, X. Yang, and M. C. Reyes, "A comparative performance analysis of data resampling methods on imbalance medical data," *IEEE Access*, vol. 9, pp. 109 960–109 975, 2021.

[23] D. A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[24] S. D. Correia, J. Fé, M. Beko, and S. Tomic, "Development of a test-bench for evaluating the embedded implementation of the improved elephant herding optimization algorithm applied to energy-based acoustic localization," *Computers*, vol. 9, no. 4, p. 87, 2020.

[25] S. D. Correia, M. Beko, L. A. Da Silva Cruz, and S. Tomic, "Implementation and validation of elephant herding optimization algorithm for acoustic localization," in *2018 26th Telecommunications Forum (TELFOR)*, 2018, pp. 1–4.

[26] J. Fé, S. D. Correia, S. Tomic, and M. Beko, "Swarm optimization for energy-based acoustic source localization: A comprehensive study," *Sensors*, vol. 22, no. 5, 2022.

[27] J. Ziv and A. Lempel, "Compression of Individual Sequences via Variable-Rate Coding," *IEEE Transactions on Information Theory*, vol. 24, no. 5, pp. 530–536, 1978.

[28] Welch, "A technique for high-performance data compression," *Computer*, vol. 17, no. 6, pp. 8–19, 1984.

[29] M. Mitzenmacher, "On the hardness of finding optimal multiple preset dictionaries," *IEEE Transactions on Information Theory*, vol. 50, no. 7, pp. 1536–1539, 2004.

[30] S. Haldar-Iversen, "Improving the text compression ratio for ascii text using a combination of dictionary coding, ascii compression, and huffman coding," Master's thesis, UiT Norges arktiske universitet, 2020.